TRINITY COLLEGE

Traffic Loop

Cian Clarke
B.A. (Mod.) Computer Science
Final Year Project April 2011
Supervisor: Prof. Vinny Cahill

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

__________________________________

Cian Clarke, April 6, 2011
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

__________________________________

Cian Clarke, April 6, 2011
Acknowledgements

Thanks to my supervisor Prof. Vinny Cahill, whose contribution and guidance throughout the project has been invaluable. Thanks to Andrew Penrose and the IBM Wikis Team for a much needed and highly enjoyable break from academia. Lastly, thanks to my family, without whom I would not have reached my final year.
Abstract

The problem of traffic mapping is not a new one, and many traffic mapping systems already exist, but most are based on a set source of input. Traffic Loop is a portable traffic mapping solution that attempts to overcome this problem, with an extensive abstraction layer and a framework based approach. It has the potential to perform on large-scale deployments, and is an easily extensible platform.

Traffic congestion is a growing problem in cities not only in Ireland, but worldwide. According to the CSO, the number of vehicles on Irish roads between the years 1988 and 2008 has grown 255%. With more and more vehicles on the road, the problem of congestion has become much greater.

Ireland has a variety of methods in use to calculate traffic density. The most popular method is the use of induction loop networks installed at traffic lights, but as cheaper more effective methods come to the fore the interoperability of these systems becomes a problem. Traffic Loop takes an abstract ‘sensor’ based approach, and can map traffic density information from a wide variety of input.
Contents

Acknowledgements ........................................................................................................................................... 4
Abstract.......................................................................................................................................................... 5
Contents .......................................................................................................................................................... 6
List of Figures................................................................................................................................................... 10
1. Introduction ................................................................................................................................................ 10
   1.1. Project Aims ......................................................................................................................................... 12
   1.2. Motivation ............................................................................................................................................ 13
   1.3. Readers Guide ...................................................................................................................................... 14
2. Background ................................................................................................................................................. 15
   2.1. Urban Traffic Control Systems ........................................................................................................... 15
   2.2. Induction Loops .................................................................................................................................. 16
      2.2.1. SCOOT ......................................................................................................................................... 16
      2.2.2. SCATS ....................................................................................................................................... 16
   2.3. Criticisms of Induction Loops ............................................................................................................. 16
   2.4. Floating Car Data ............................................................................................................................... 17
   2.5. Camera-Based Systems ....................................................................................................................... 17
   2.6. Traffic Mapping in Ireland .................................................................................................................. 17
      2.6.1. Dublin City Council .................................................................................................................... 17
      2.6.2. RDS-TMC ................................................................................................................................... 18
      2.6.3. AA Roadwatch ............................................................................................................................ 18
      2.6.4. NRA Traffic ............................................................................................................................... 19
2.6.5. Real-Time Display of Dublin Traffic Information on the Web .................................. 19
2.6.6. Using Vehicle Context Information to improve intelligent traffic management ....... 20

2.7. Traffic Mapping Worldwide ......................................................................................... 20

2.7.1. Google Maps ........................................................................................................... 20
2.7.2. Bing Maps Traffic .................................................................................................. 21
2.7.3. GATI ......................................................................................................................... 22
2.7.4. Houston TransStar Traffic Map ............................................................................ 24
2.7.5. Dallas / Fort Worth Traffic ...................................................................................... 24
2.7.6. San Diego Real-Time Traffic .................................................................................. 25

3. Design .............................................................................................................................. 26

3.1. Feature Analysis .......................................................................................................... 26

3.1.1. Congestion Lines .................................................................................................... 26
3.1.2. A Framework ......................................................................................................... 26
3.1.3. Abstract from technology ....................................................................................... 26
3.1.4. Ability to pan and zoom .......................................................................................... 27
3.1.5. Traffic Accident and Roadwork Mapping, Lane Closures ..................................... 27

3.2. Abstraction .................................................................................................................. 27

3.2.1. Abstraction Specification ........................................................................................ 28
3.2.2. Sensor – An abstract concept of input .................................................................... 28
3.2.3. A ‘UTC Proxy’ ....................................................................................................... 29
3.2.4. Modules .................................................................................................................... 29

3.3. Technologies Used ....................................................................................................... 32

3.3.1. Python ....................................................................................................................... 32
3.3.2. PHP ........................................................................................................................... 32
3.3.3. MySQL ..................................................................................................................... 33
3.3.4. JavaScript ............................................................................................................... 33
3.3.5. HTML & CSS ......................................................................................................... 33
3.3.6. Apache & the LAMP/MAMP Stack ......................................................................... 33
6.3.5. Public Transport Operators ................................................................. 56
6.3.6. Commuters ....................................................................................... 56
6.3.7. Environmental Benefits ..................................................................... 56

7. Conclusions ............................................................................................... 57

7.1. Expanding upon Traffic Loop ................................................................. 57
  7.1.1. Twitter Sentiment Analysis ............................................................... 57
  7.1.2. Computer Vision ............................................................................. 57
  7.1.3. Public Transport Probes ................................................................. 58
  7.1.4. Data Visualisation .......................................................................... 58
  7.1.5. Native Mobile Application ............................................................. 58

7.2. Criticisms ............................................................................................... 58

7.3. Conclusions ............................................................................................. 59

8. Appendix .................................................................................................... 61

8.1. Appendix A: Sequence Diagram ............................................................ 61

8.2. Appendix B: Code & Website ................................................................. 63

9. Bibliography ............................................................................................... 64
List of Figures

Figure 1 Dublin City Traffic Cameras........................................................................................................18
Figure 2 – AA Roadwatch Map......................................................................................................................19
Figure 3 - Google Maps UK Traffic Information..........................................................................................21
Figure 4 - Bing Maps Traffic, Seattle Area – Note partial coverage of the City of Bellevue, as mentioned below..................................................................................................................22
Figure 5 - City of Bellevue real-time traffic map .........................................................................................23
Figure 6 - Google Arterial Traffic Information............................................................................................23
Figure 7 - Houston TransStar Map ..............................................................................................................24
Figure 8 - Dallas / Fort Worth Traffic ........................................................................................................25
Figure 9 - Virtual Sensor Mapping.............................................................................................................31
Figure 10 - Architecture Overview ............................................................................................................35
Figure 11 - UTC Proxy with SCOOT UTC Shown .......................................................................................36
Figure 12 - PHP Application......................................................................................................................36
Figure 13 - Diagram of API databases - key values italic with key icon......................................................40
Figure 14 - The Administrative Panel........................................................................................................41
Figure 15 - Frontend ..................................................................................................................................42
Figure 16 - Routes Panel............................................................................................................................43
Figure 17 - Sensor Marker ..........................................................................................................................43
Figure 18 - History Slider Control................................................................................................................43
Figure 19 - Congestion Colour Legend .........................................................................................................44
Figure 20 - Traffic Loop UI..........................................................................................................................44
Figure 21 - Detail view of some congestion information ........................................ 45
Figure 22 - Edit Cities Panel .................................................................................. 50
Figure 23 - Edit Routes Panel .................................................................................. 50
Figure 24 – Edit Sensors Panel ................................................................................ 51
Figure 25 - Options Panel ......................................................................................... 52
Introduction

Traffic congestion is not a new problem – in fact, it is an issue that existed since the era of the horse and cart. Only with the advent of the motorcar, however, did the issues of gridlock and widespread congestion issues come about.

Between the years 1988 and 2008, the number of cars on the road increased by 1,516,272\(^1\) vehicles, a 2.5 times increase.

Needless to say, this increase has resulted in the need to increase the capacity of the road network. Whilst these capacity increases are possible with peripheral routes, there is little that can be done to increase capacity in historic city centres where the ability to expand is often limited or non-existent.

The ability to monitor traffic as it progresses throughout the day is an invaluable asset. Present solutions make use of manual user input from traffic controllers observing the roadway on mounted cameras along with the input given by induction loop networks throughout our city centres. Most traffic mapping solutions focus on one particular city or area. No open & freely distributed solution was discovered that is able to take input from a number of sources, automatically collate and combine into density information for roadways without the need for user input.

1.1. Project Aims

The goal of this project is to provide a traffic-mapping framework. The solution should be capable of taking multiple sensor sources as input, collate this data and map a value of congestion to a uniform scale on a roadway. This should be achieved without the need for ongoing human feedback. This frees up traffic controllers from having to manually monitor the traffic situation on major routes, and gives more time to focus on the greater citywide picture, enabling a much more coordinated approach to managing traffic.

One of the most important parts of the implementation is that a degree of abstraction exists between the monitoring and reporting client and the modules that it uses to retrieve data. This
would mean the user interface and application logic would have no concept of where the data is coming from beyond the fact that it is a ‘sensor’, with a certain value.

Another important consideration is how often the reporting client is polled. In the event the mapping user interface is a public facing website, there is a strong possibility the site will see a usage spike before commuters begin their journey to and from work at 8am and 4.30pm. Polling the traffic system constantly for new data could cause the system to become unresponsive, an outcome with dire consequences. Rather than constantly polling, a local replica is required which polls for new information on a user defined schedule.

The system should be capable of dealing with spikes of interest from commuters viewing the traffic map prior to setting off on their morning commute and on their return journey home.

The solution should also be portable, and not be dependant on the architecture available in one particular country or continent. It should be capable of being implemented in any city worldwide.

1.2. Motivation

A large portion of the motivation for this project is driven by the quantity of induction loop sensors installed in Irish roadways, and a desire to put these sensors to greater use.

As new technology emerges, however, and more modern traffic monitoring solutions come to exist, there is also a risk that legacy systems will not co-exist in harmony with newer monitoring solutions. To this end, the need for a framework abstract from the underlying technology was devised.

Lastly, a part of the motivation is through a personal gripe of the author. Although sensors capable of measuring congestion exist, traffic-mapping facilities are frustratingly primitive when compared to solutions prevalent in the United States. From the point of view of commuters, the only current means of monitoring congestion, on even the country’s busiest of routes, is listening to radio traffic reports. A mapping solution would be a vastly improved facility for commuters.
1.3. Readers Guide

Background

This chapter introduces the problem of traffic monitoring and demonstrates the need for an open solution. An examination of the solutions to traffic mapping both in Ireland and across the globe illustrates any prior art that exists in this field.

Design

The design of the project is discussed. First, based on systems pre-existing, a feature analysis is performed where the essential components of a traffic-mapping framework were decided. Then, the approach to abstraction is discussed. The technologies decided upon are outlined.

Implementation

The architectural components of the solution are discussed: The python proxy, the PHP application, the MySQL database, the administrative panel and the frontend. A sequence diagram is presented.

Documentation

The installation of the framework is dealt with. Instructions are also provided for setting up Traffic Loop, writing a UTC module and adding new cities.

Evaluation

Traffic loop is evaluated based on portability and performance, and the benefits the system brings are presented.

Conclusions

Future capacity for expansion is discussed, along with criticisms of the project and a conclusion.
2. Background

In this chapter, we examine the problems of traffic in Ireland. We cover pre-existing traffic monitoring systems in use in Ireland and internationally, and the limitations associated with these.

Introduction to Traffic in Ireland

As presented in the introduction, traffic congestion is a growing concern. CSO statistics indicate Ireland in 1988 had 981,296 vehicles on the road. Contrast this with 2,497,568 in 2008, and we see a worrying trend developing. In this same year there were 2,632,136\(^2\) licensed drivers on Irish roads, who travelled an average of 16,376 kilometres.

There is also a greater length of paved roadway to cope with – even between the years of 2005 and 2008 we see an increase of 171\(^%\) on the length of paved motorway\(^3\).

Although statistics were not available for Ireland, traffic congestion was expected to cost 1\(^%\) of the European Union’s GDP in 2010\(^4\).

While it is apparent that there is an increasing demand being placed on Ireland’s infrastructure with increasing volumes, what are the systems in place to manage this?

2.1. Urban Traffic Control Systems

An Urban Traffic Control System (UTC herein) is a subset of an Intelligent Transport System that aims to alleviate the problems of traffic congestion by some form of intelligent sensor based means. These systems are typically used to control a traffic light network and implement bus or public transport priority and to assist emergency vehicles travelling on emergency callout through a city. We are primarily concerned with the sensing capabilities of such systems, particularly in measuring traffic density.
2.2. Induction Loops

The primary means of controlling traffic across the globe is through the use of induction loop sensors installed in the roadway. An induction loop is a magnetic coil installed in the roadway that can detect the presence of a vehicle. By using a network of these inductive loops a UTC system can determine the mean flow travelling over the sensor. This is then used to determine the optimum cycle time, or green time given to a junction.

2.2.1. SCOOT

Scoot (Split Cycle Offset Optimization Technique) is one of the world’s most popular induction loop based UTC systems, with deployments in major world cities such as Beijing, Bangkok, and London. Its development is lead by Siemens under the corporation name Peek Traffic Limited. Cork City received its installation of the SCOOT UTC in 1997, and Galway City is considering the installation. In addition, Limerick City also has an already active installation of SCOOT.

In addition to flow counts, SCOOT is capable of outputting a saturation measurement for traffic congestion.

2.2.2. SCATS

The SCATS (Sydney Coordinated Adaptive Traffic System) is another induction loop based UTC originally developed by the Roads and Traffic Authority of New South Wales, Australia. SCATS has existed since 1976, and has been implemented across the globe with installations located in excess of 145 cities. In Ireland, SCATS is in use in Dublin, Waterford and Wexford. Researchers found that in 1996, the Dublin SCATS system only utilised 170 of Dublin’s 450 signal controllers – it is possible this has extended since.

2.3. Criticisms of Induction Loops

The cost of installing an induction loop based UTC system can be quite expensive. One recent quotation was the outlay of the Scoot UTC Milton Keynes. Excluding their car park monitoring solution, total outlay was in the region of £1.3 million.
Another publication puts the cost at £20,000 to £25,000 per junction\textsuperscript{16}, a massive outlay. Not only are induction loops expensive to install, the costs of maintenance are also extensive. As the loop is buried in the roadway, repairs require excavation. As more modern congestion measuring solutions come to the fore, induction loop technology is likely to be replaced by cheaper and easier to maintain technology that does not require installation under the roadway.

2.4. Floating Car Data

Floating Car Data is a method of using on-board sensor data along with cellular phones travelling in vehicles as a sort of floating sensor to determine the speed of travel along a roadway. This information can be collected by vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications, and can potentially be used as the source of a measure of congestion on a roadway. A major benefit to Floating Car Data is the lack of a need for any type of sensor present on the roadway, which reduces the cost of such a solution.

2.5. Camera-Based Systems

Both stills and video cameras can also be used to obtain a traffic reading from a roadway. Traffic controllers frequently use a network of mounted traffic cameras to manually determine the level of congestion on a roadway. Using computer vision techniques such as feature tracking, an accurate estimate of traffic can be obtained automatically\textsuperscript{17}.

2.6. Traffic Mapping in Ireland

2.6.1. Dublin City Council

Dublin City Council also monitors traffic using its network of mounted cameras throughout the city\textsuperscript{18}. 
From this information along with collected data from listeners via Twitter\textsuperscript{19} and SMS message, the Dublin City Council Traffic Control Centre broadcasts ‘Live Drive’, a traffic update service on Dublin City FM\textsuperscript{20}.

2.6.2. RDS-TMC

Dublin City Council is publishing traffic information across the FM band of RDS-TMC, broadcast by RTÉ\textsuperscript{21}. This system provides traffic incident information to satellite navigation units (both built into automobiles and set-top units). The unit can then offer to re-route the traveller to their destination, avoiding the incident.

2.6.3. AA Roadwatch

AA Roadwatch is one of the primary providers of traffic information nationwide\textsuperscript{22}, with extensive coverage of road works and incidents that have occurred displayed on a map. These reports are also broadcast on national radio during news reports\textsuperscript{23}. AA Roadwatch tends to focus on specific incidents that already have occurred rather than focusing on regular congestion. Its input comes largely from reports from county councils, Gardaí and the public rather than from sensor data.

The closest implementation to a visual display of traffic information in Ireland is the map provided by AA Roadwatch displaying incidents nationwide.
Whilst this is an invaluable resource, it only displays traffic black spots. Incidents that are not severe enough to be reported by commuters or spotted by their traffic monitoring helicopter services can go undetected.

2.6.4. NRA Traffic

The National Roads Authority (NRA) run a website for traffic information, http://www.nratraffic.ie. The map interface on this site does not allow the map to be dragged – pan and zoom buttons have to be used, which makes navigating very difficult. The camera plots the locations of traffic cameras – clicking an icon displays the traffic camera. The website has attempted to implement journey times as a measure of congestion, however at the time of writing this was part operational on the M50 and M1 roadways only.

2.6.5. Real-Time Display of Dublin Traffic Information on the Web\textsuperscript{24}

A mapping approach was taken by a Trinity College masters student to map traffic data coming from SCATS in Dublin City, but the focus of this project did not feature any extensive layer of abstraction. A lot of the project involved dealing with data from SCATS and determining a figure of saturation from this data. As a result, the solution was tied to the SCATS UTC.

In addition, this project was conceived in the year 2000, when there was a distinct lack of
easy to use mapping API’s to interface with on the web. Google Maps was not even announced until 2005\textsuperscript{25}.

2.6.6. Using Vehicle Context Information to improve intelligent traffic management

An interesting piece of research studying the effect of providing passengers travelling via public transport access to real-time route information suggests that by providing travel time information, it could promote the use of public transport and reduce the number of vehicles on the road (Fagan, Meir. Using Context and Behavioural Patterns for Intelligent Traffic Management)\textsuperscript{26}. This could also suggest that the introduction of a congestion-mapping framework would bring similar benefits.

2.7. Traffic Mapping Worldwide

We will now examine the traffic solutions being deployed worldwide, to gain a greater scope and understanding of the solutions that have been successful elsewhere.

A large bulk of the notable traffic measuring systems are deployed in the United States, where a very large network of induction loop based systems exist. Interestingly however, the United States makes use of traffic cameras extensively also. Compared to the rest of the world, the United States has a huge portion of it’s roadways covered by traffic control centres – one possible reason for this has been the justification of funding such systems as a benefit to homeland security\textsuperscript{27}.

2.7.1. Google Maps

Perhaps the most prevalent traffic mapping resource to emerge has been Google Maps Traffic Information that emerged in 2007\textsuperscript{28}. Google Traffic is currently available in Canada, China*, Finland, Hong Kong, Italy, Russia*, Singapore, Thailand, Ukraine, United Kingdom* and the USA\textsuperscript{29}. This information in the USA is collected by a combination of US Department Of Transport induction loop sensors in the roadway and crowdsourcing congestion data\textsuperscript{30}. In the UK, the data is sourced from the UK Highways Agency\textsuperscript{31}. 20
Although Google Traffic is a very effective solution for mapping congestion, its rollout has not been widespread. In the United Kingdom it only exists in major cities, and in Ireland there is no coverage at all. Their traffic system is closed – there is no API allowing for input and mapping of new data. In addition, a small city or town is unlikely to approach Google looking to input their data. The system is closed, with little or no information available about it’s implementation or the source of the data.

2.7.2. Bing Maps Traffic

Bing Maps have taken a very similar approach as Google to traffic mapping, displaying coloured lines to represent the density of traffic on a roadway. A technology called Microsoft Clearflow is being used to show this data, applying clever machine learning to calculate traffic\textsuperscript{32}. At the time of writing, only cities in the US and Canada are supported.
2.7.3. GATI$^{33}$

GATI, or Google Arterial Traffic Information is a University of Washington project aiming to map traffic information focusing on not just freeway routes, but also arterial roadways.

The problem with the pre-existing map being used by the city of Bellevue was scale. Roadways shown on the map were not to scale, and did not represent the length of some streets. This can lead to much confusion amongst commuters, and makes the map difficult to decipher.
Figure 5 - City of Bellevue real-time traffic map

This system used a Google Map which can be scaled and panned, and plots the traffic on this map using coloured lines. This system does not, however, make use of a layer of abstraction as is proposed here.

At the time of writing, the last posted update on the GATI project is from 2008. The project is not operational on the City of Bellevue website, and no working example with traffic displayed on the map could be located.

Figure 6 - Google Arterial Traffic Information
As we see from the figure above, the user interface is intrusive and far from elegant. The quantity of traffic segments could possibly lead to confusion. Perhaps development has stalled since the advent of Google Maps Traffic and Bing Traffic, however these systems only plot the major freeway routes through the town.

2.7.4. Houston TransStar Traffic Map\textsuperscript{34}

The Houston TransStar Traffic Map is a static traffic map for the city of Houston. Although the default map cannot be zoomed or panned with ease, an optional alternative provided by Microsoft Bing Maps is provided. The map is highly feature rich, and allows not only colour aided mapping of congestion, but also incidents, construction, traffic cameras, weather information and many more useful markers.

![Figure 7 - Houston TransStar Map](image)

2.7.5. Dallas / Fort Worth Traffic\textsuperscript{35}

The Dallas / Fort Worth traffic map is remarkably similar to the Houston TransStar Traffic Map. The mapping engine used for this is a Microsoft Bing Map. Interestingly, the default view does not show any useful traffic information – the option for traffic speeds must be checked using the UI panel provided on the left of the screen. This unobvious action leads to a somewhat poor user experience – there is a possible risk many users visiting the site will not see this very small control.
The real time traffic map deployed by San Diego is a static map view with coloured traffic information drawn upon it. While the map cannot be panned, zoomed, scaled or filtered, a very useful panel is shown to the right of the map showing the average speed of vehicles gathered by detectors on various routes.
3. Design

In this chapter, we examine the implementation of a mapping system capable of taking input from any type of traffic sensor and displaying the data in a meaningful manner. We first examine the features decided upon, then the architecture of the system. Finally, we examine the specification made to ensure abstraction, and the process that went into designing a system that complied.

3.1. Feature Analysis

Largely based on the features identified in the traffic monitoring solutions discovered above, a feature analysis was conducted to determine what would be required to make this system most useful.

3.1.1. Congestion Lines

At the fore of any traffic mapping system is its ability to draw the congestion value on a map. The most common and effective approach displayed in all prior examples is drawing a coloured line on the roadway to indicate congestion. Such a line enables commuters to follow their route from origin to destination on a map, and see the degree of congestion throughout. Such congestion lines were determined as a core requirement.

3.1.2. A Framework

The scope of this project was not just to map the congestion of the city of Cork. Rather, this traffic mapping system was to be provided as a framework, which can be deployed in any city worldwide. Through examining other solutions worldwide, a clear lack of such a framework existed. It is hoped a framework would benefit commuters and city councils alike to plot areas that had not previously been covered by existing congestion mapping solutions.
3.1.3. Abstract from technology

Unlike all of the examples encountered previously, this framework was to be abstract from an intrinsic tie to a particular type of traffic data or sensor. A modular based approach depending on the input was to be adopted – this is discussed at length under the title ‘abstraction’.

3.1.4. Ability to pan and zoom

A large limitation of many previous mapping systems in place was the inability to drag the map to pan, and often the ability to zoom was not implemented at all. As a result, a feature rich mapping API was the chosen approach rather than a static image of a map, drawing congestion upon it.

3.1.5. Traffic Accident and Roadwork Mapping, Lane Closures

In addition to congestion mapping, some of the more notable previous examples added to their system the ability to map incidents that have occurred such as traffic accidents, road works or lane closures. Another common feature was the ability to display a marker on the map where a traffic camera or overhead display exists.

None of these features lend themselves well to an abstract based approach. Not all hardware is capable of storing the fact that a lane has been closed, or that there has been an accident. Rather, the main scope of the project was to ensure that a robust and abstract congestion-mapping framework was built. Additional features such as traffic accident and roadwork location mapping could be deployed on top of the existing User Interface for the individual cities implementing the framework.

3.2. Abstraction

Very early on in the project, the important issue of abstraction arose. Simply implementing a traffic measuring system for the SCOOT UTC would perhaps benefit Cork or Limerick, but not every Irish city (or indeed cities worldwide) could make use of this. The ideal solution would allow modular expansion from other UTC systems.

In addition, as we’ve already seen, UTC systems are expensive to install, repair and maintain. As Computer Vision and Floating Car Data techniques advance, the need may arise to add
input from another sensor type, a system far detached from the induction loop & junction weighted method used in an UTC system.

The problem with the proprietary nature of UTC systems, and a means of data exchange between each stage of the UTC system (Dineen & Cahill, Towards An Open Architecture For Real-Time Traffic Information Management, 2002) has already been examined. Ultimately, however, this representation was deemed to contain data too verbose and beyond the interest of our traffic mapping framework.

A strict specification of abstraction was devised in order to ensure that at all stages of development, there was no reliance on one particular format of input. This specification follows.

3.2.1. Abstraction Specification

The main purpose of the abstraction layer is to make plug and play additions of different traffic control systems or sensors an easy task. Although the main focus of the project was initially to take input from the SCOOT UTC (a detector based source), the system needs to be capable of accepting input from a multitude of other sources.

As a result, a generic term to refer to any source of input was devised – ‘Sensor’.

3.2.2. Sensor – An abstract concept of input

In constructing the concept of a sensor, a list of potential input sources was devised:

- Induction Loops from a UTC
- Mounted Traffic Cameras
- Speed Cameras
- Natural Language Processing on traffic reports / user feedback
- Twitter searching for geo-tagged tweets mentioning traffic

The common attributes of each were examined. The three most important attributes determined were:

- **Sensor ID**

  *A unique identifier which every sensor, be it a junction detector or a camera, should have & be polled from.*
• **Timestamp**

_The time the data was recorded at. Interval can be determined by examining the previous timestamp for that particular Sensor ID._

• **Sensor Value**

_An established 0-100 scale to determine the level of saturation for this particular sensor. If this were vehicle flow per hour, a degree of saturation could be determined for this._

_Many UTCs already output this data in saturation form on a 0-100 scale._

**JSON Example:**

```json
{
  "sensor": {
    "id": "DC122A1",
    "timestamp": "YYYY-MM-DD HH:MM:SS",
    "value": "75"
  }
}
```

### 3.2.3. A ‘UTC Proxy’

The proxy will simply be polled for a list of sensors, and will not be concerned with piecing this data together. This task will be left to the application performing the request, and it will piece the data together from black box ‘sensors’, the type of which it is unaware.

As a result, the application could be polling data for just one route, or all routes capable of being plotted. No overlap of redundant data can be returned.

The implementing application will have a mapping of combinations of sensor IDs to routes stored in a database.

### 3.2.4. Modules

All of the following modules as described are outside of the abstraction layer – that is, they implement sensor input for a specific hardware.

#### 3.2.4.1. The SCOOT / Astrid Proxy Module

The Astrid proxy is used to retrieve the data required by the ‘sensor’ model mentioned above from the Astrid UTC database. The implementation retrieves traffic information from
plaintext log files output by the UTC. A description of how the above common attributes applies to this specific module follows:

**Sensor ID:** *The unique ID of the detector in the SCOOT network. See note on Virtual Sensors.*

**Timestamp:** *Information available directly from the log files. Duration can be determined by checking the next chronological entry for the relevant sensor ID.*

**Sensor Value:** *The ‘saturation’ field of the ASTRID log files should prove sufficient for determining the value required. Most of the logic to normalize and calculate an accurate value has already been done by the system, and should produce a valuable calculation of traffic.*

*If this figure were not available, the value could be determined by analysing the vehicle flow count.*

‘**Virtual Sensors**’: Often, multiple detectors make up one direction in a junction – this necessitates the building of ‘virtual detectors’.

The value of these virtual detectors is calculated by average weighted mean of a number of physical detectors installed in the roadway, mapping to one distinct value.

Virtual detectors span a number of lanes in one specific direction on a roadway.

To map physical sensors to one virtual sensor, a database of virtual to physical mappings seemed like the optimum solution. However, any database interaction performed in SCOOT required an installation of that particular database software on the same machine. This was determined an excessive overhead. Instead, a CSV file mapping a virtual sensor ID to a number of physical sensors was established. This simple approach allows for the mapping of sensors by even a non-technical user – any modern spreadsheet software can export a CSV file. The CSV file contains a virtual sensor name, followed by a list of physical sensors and their weighting. This need for a weighting is introduced because very often a lane used for turning left has less importance to the congestion of a roadway than central lanes. CSV files take the format:

```
virtualSensorName, physicalSensor1, weightSensor1, physicalSensor2, weightSensor2
```

An example listing for a junction follows:
The end congestion figure is calculated as follows:

\[
\frac{wT_1 \cdot l_1 + wT_2 \cdot l_2 + wT_3 \cdot l_3 + wT_4 \cdot l_4}{\text{sum}(wT)}
\]

Figure 9 - Virtual Sensor Mapping

3.2.4.2. A Traffic Camera based proxy module

This sensor is a hypothetical construct used to ensure the concept of ‘sensor’ is truly abstract from the type of sensor being input.

The traffic cameras in question are mounted on top of traffic lights, facing into traffic towards the cars as they wait. To get a value of saturation, the camera is polled in real time.

Each camera has a sensor ID, and is used to make up a route. The camera returns a saturation value, which could be calculated using optical flow or feature tracking to determine the number of vehicles on the roadway.

**Sensor ID:** *Each camera has it’s own unique ID*
3.3. Technologies Used

We will now examine the technologies decided upon for this project. When deciding, the brief was simple - open source alternatives were to be given preference, and if at all possible, free.

Cross-platform solutions were preferable; no tie to one particular operating system should exist for client or server.

3.3.1. Python

The proxy was written in Python largely due to the requirement that it run on multiple hardware platforms with low setup cost. Many Linux & Macintosh distributions include Python by default, and an installer exists for Microsoft Windows. Another reason for its choice was the relatively low barrier to entry when learning the language. The author had no prior experience with Python, and came across no difficulties in implementing a non-trivial software solution in the language.

3.3.2. PHP

PHP is a scripting language which is well suited to web development, and this project in particular because of it’s close ties to the database technology MySQL. PHP is a free scripting language, and is well suited to the server side processing of web application development.

Because PHP scripts run as a file on a web server, this leads to the by-product that a PHP application can be used as an API for other software, which means no intrinsic link to the frontend would exist.
3.3.3. MySQL

MySQL is a relational database management system also released free under the GNU GPL licence. Reasons for choosing this technology are twofold – databases can easily be polled by PHP applications, and can also very easily be manipulated by a database frontend, ‘phpMyAdmin’.

3.3.4. JavaScript

JavaScript was chosen as the primary technology for rendering the web application. Websites with a high public exposure such as Google’s Maps and Gmail, and Facebook have helped bring JavaScript web applications to the fore in recent times. The ability to dynamically load new content without a page refresh suited the project, and the author’s high familiarity with building dynamic web applications made it an appealing technology of choice. Although the frontend is structured in HTML, most of the content is inserted dynamically and asynchronously. This means the content can be refreshed on a regular (3 minute) basis, and altered on the fly.

Two additional JavaScript libraries were also used. The JQuery library and Douglas Crockford’s JSONify utility.

3.3.5. HTML & CSS

HTML was chosen to structure the frame of the web application. Although the majority of the user interface is a map, some HTML ‘div’ elements were required for menus and panels. The administrative interface is also structured using HTML. Dynamic elements of the page such as timestamps are then inserted using JavaScript once the required resource has loaded.

3.3.6. Apache & the LAMP/MAMP Stack

The first letter in the AMP stack denotes a webserver on a Linux (LAMP) or Macintosh (MAMP) operating system. The remainder of the letters, ‘AMP’ indicates the server runs Apache, PHP and MySQL. We have already discussed the use of PHP and MySQL, but it is worth examining the benefit that the use of the Apache webserver will bring to this project.
There is no intrinsic tie to Apache as the webserver of choice – in fact, the only apache-specific technology supplied is an ‘.htaccess’ file supplied as a basic solution to secure the administrative section. It is the recommended server of choice, however.

A number of reasons exist for this choice. Firstly the installation of the LAMP stack with any modern Linux operating system is straightforward, if it is not already bundled.

Apache is the webserver of choice for most shared hosting environments, and if running on a Virtual Private Server, the scalability of the application can typically be delegated to the system administration staff of the hosting company.

The optimisation of Apache to scale thus meeting high demand is simple, even for those with little to no system administration experience. Even the simple step of enabling the ZEND Optimizer can increase PHP page load speed by up to 50%\textsuperscript{40}.

3.3.7. Google Maps

Google Maps was chosen as the mapping API for this project because of its comprehensive coverage of Irish roadways. It has been the leading provider of online mapping since it’s launch in 2005, and the only main competitor is Bing Maps.
4. Implementation

4.1. Architecture

The architecture of Traffic Loop is that of three separate entities.

- A Python webserver to act as a proxy between the UTC and the PHP application.
- A PHP application to poll the traffic system proxy, which interfaces with a MySQL Database that stores a local database of traffic information
- An AJAX driven frontend user interface consisting of a Google Maps view with custom interface elements.

![Architecture Overview](image)

Figure 10 - Architecture Overview

4.1.1. The Python Proxy

The Python proxy is built so that it can sit on a remote machine and be polled by the PHP Application. The proxy could sit on a UTC computer accessing local log files, or a camera multiplexer querying the feeds incoming. The only requirement on the hardware is its ability to run Python 2.6, and as a result the proxy could run on a Windows, Unix or Macintosh operating system.
The proxy is built to be lightweight as there is a possibility the client will be placed on mission critical hardware. In addition, requests to the Python proxy are minimized to reduce load. In addition to the cross-platform nature of the language, the computer vision framework OpenCV is available in Python, making computer vision based approaches to sensor-based input perfectly suited.

Proxy modules are written as python based modules. A module is added to the system by simply copying the Sample.py format and filling in the method stubs as described in ‘Writing a UTC Proxy Module in Python’.

4.1.2. The PHP Application

The PHP application accepts requests from the frontend and serves up the requested routes to be plotted on a map by the Ajax application.

Figure 11 - UTC Proxy with SCOOT UTC Shown

Figure 12 - PHP Application
It first polls a local replica of traffic information in a MySQL database. If recent enough traffic information can be retrieved from the database (to within a customizable threshold, default 3 minutes) the local replica is used. Otherwise, the UTC proxy is polled for more recent traffic information, and the database updated.

In constructing the routes, the PHP application first queries the database for either the specific routes requested, or all routes. This then gives the application an array of routes objects containing a name, direction and array of sensor IDs.

Each sensor ID associated with that particular route is iterated over, and it’s coordinates looked up on the sensors database. This means each sensor that makes up a route now has a position attached to it.

Lastly, each sensor’s traffic congestion value is built. This is achieved by polling the database for the traffic value of the current sensor. This is averaged over time by use of a simple database query:

```sql
SELECT AVG(value) from traffic where timestamp>='$dateString' AND id='$id' limit 10
```

Here, $dateString is a timestamp from 15 minutes ago and $id is the sensor id being requested. In addition to normalizing over time, the values are also normalized over space to ensure the value for one sensor does not spike unnecessarily. The data for one particular sensor is a mean of its own value and a 50% weighted average of the previous and the next sensor value along the route.

The PHP application returns a JSON object, meaning that there is no tight coupling between the PHP application and the AJAX frontend. The PHP application can be treated as an entity of its own which functions as an API to Traffic Loop. JSON parsing libraries exist in most major programming languages. This enables a native iPhone or Android mobile application to be built with ease to display the information, or indeed a desktop client.

The API can be polled using POST or GET. The routes attribute means the API can be polled for just one specific route – if left blank, the default return value of all available routes is returned.

The attribute ‘historic’ can also be set to an integer number, which returns for each sensor in a route, the requested number of values. This illustrates a historic context of how traffic has evolved through the day. This defaults to false, but the frontend implemented makes use of this feature.
The API returns a timestamp used to show the mean time that the data applies to, along with a routes object and a log string to which errors are appended.

A simplified version of what the API returns for one route with two historic values follows:

```json
{
    "time": "12:42:48",
    "routes": [
        {
            "name": "N8",
            "sensors": [
                {
                    "id": "DC117A1",
                    "timestamp": "2011-03-21 12:10:37",
                    "value": "54",
                    "latitude": "51.9062",
                    "longitude": "-8.39911"
                },
                {
                    "id": "DC117A1",
                    "timestamp": "2011-03-14 17:17:11",
                    "value": "60",
                    "latitude": "51.9062",
                    "longitude": "-8.39911"
                }
            ],
            "direction": "0"
        },
        {
            "id": "DC118A1",
            "timestamp": "2011-03-21 12:10:37",
            "value": "35",
            "latitude": "51.9028",
            "longitude": "-8.4307"
        },
        {
            "id": "DC118A1",
            "timestamp": "2011-03-14 17:17:11",
            "value": "60",
            "latitude": "51.9028",
            "longitude": "-8.4307"
        }
    ],
    "log": ""
}
```

This is the N8 route, which is made up of two sensors – DC117A1 with 54% congestion and DC118A1 with 60% congestion.

A simplified JavaScript implementation of the parsing of this information follows:
foreach (json.routes as route){
    var routeName = route.name;
    for (var i=0; i<sensors.length-1; i++){
        // Current and next sensor. Increment [0] for historic
        var current = sensors[i][0];
        var next = sensors[i+1][0];
        var startPos = new coordinate(current.latitude, current.longitude);
        var endPos = new coordinate(next.latitude, next.longitude);
        var trafficValue = (current.value+next.value)/2;
        drawLine(startPos, endPos, trafficValue);
    }
}

We see that for each sensor, there is an associated array of older historic values. If we have no interest in historic values, we just look at the value for index zero.

Even if no historic data is requested from the API, each individual sensor of a route is an array with one object – the sensor must be accessed with an array index sensors[i][0]. This means implementations can be consistent – the draw method for historic and current data would not differ.

We will now examine the files that make up the PHP application:

- config.php stores configuration options for the database connection and URL of the UTC Proxy, as detailed in the setup instructions of the documentation section.
- trafficloop.php contains the remainder of the PHP application. It is this PHP file that is polled to retrieve traffic information.

### 4.1.3. The MySQL Database

The MySQL database stores a local replica of traffic information to reduce the number of calls required to the UTC Proxy. It also stores sensor to GPS coordinate mappings, and route information.

The database consists of the following five tables:

The ‘routes’ table stores roadways, which are described by their name (“N8”, N27”), their direction (inbound or outbound) and a comma separated list of the sensors which map this route.

The ‘traffic’ table stores a list of unique sensor IDs, a value between 0 and 100 of traffic saturation, and a timestamp for when this data was most recently recorded for.
The ‘sensors’ table stores the positions of sensors on a map. This is described by a list of unique sensor IDs and a latitude and longitude value.

In addition, there is a ‘cities’ table that stores the list of cities the UI can pan to. Routes are not linked to cities, so populating any more than one city is optional. The table can be left empty if just utilising the PHP application as an API.

A ‘routesCache’ table exists, which the frontend utilises to cache its route coordinates, reducing calls to the Google Maps Direction Service. This table does not require any manipulation by the end user.

![Diagram of API databases - key values italic with key icon](image)

**Figure 13** - Diagram of API databases - key values italic with key icon

### 4.1.4. The Administrative Panel

The administrative panel was built in a combination of PHP and structuring HTML. Its purpose is to act as a frontend to the MySQL database, allowing manipulation of the aforementioned Routes, Traffic, Sensors and Cities tables. A guide to using the administrative panel exists in the Documentation section.

The Administrative Panel consists of the following:

- **db.php** – database settings & connection code
- **index.php** – the tab container and tab switching JavaScript code, along with PHP includes for the relevant tabs.
The rest of the administrative codebase adopts a Model – View – Controller (MVC) organisation of sorts. Taking the example of the route tab:

**route.php** is the model.

**routeTab.php** is the view, along with database retrieval logic.

**routePost.php** contains most of the controller logic – database insertion & deletion etc.

The other tabs are cities, sensors and UI options – some of which do not require a model class.

| Cities | Routes | Sensors | UI Options |

**Administration**

**Edit Cities**

![City Name Latitude Longitude](image)

<table>
<thead>
<tr>
<th>City Name</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Edit</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dublin</td>
<td>53.3466</td>
<td>-6.26728</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Cork</td>
<td>51.8972</td>
<td>-8.46037</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Galway</td>
<td>53.2791</td>
<td>-9.0519</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Limerick</td>
<td>52.6684</td>
<td>-8.62652</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Waterford</td>
<td>52.2616</td>
<td>-7.1286</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

Figure 14 - The Administrative Panel

4.1.5. The Frontend

The frontend is exposed as a web page that features primarily a Google Maps view. Around the Google Maps view a custom user interface is built consisting of a menu with configurable options.
Figure 15 - Frontend

Options exist to turn the display of routes in both directions on and off, and to show the individual sensors that the congestion data is coming from on the map. In addition, a slider is shown which allows the user to scale through time showing previous traffic information.

On page load, the frontend constructs an AJAX POST to the PHP application requesting all routes with historic data. It also requests the latest route caches from a separate database.

When the traffic information has been returned as a JSON object, there exists an array of routes that can be plotted. Each of these routes objects has an array of sensors, which in turn have an array of readings.

Now that a picture of the routes exists, the lines that make up points between sensors must be drawn. If this is the first time we’ve seen this route, the Google Maps directions service is polled to request the line segment for this portion of the route, and the JSON to represent this is stored in the database indexed by a unique ID.

If this route has been drawn before, its line segment is simply retrieved from the database.

The UI also consists of some options panels to change configuration options.

The cities available to the UTC are drawn from a database list that contains their name and the GPS coordinates of their centre – a description of editing these can be found in the documentation.

The routes panel allows users to toggle on and off various routes drawn on the map. This table is drawn by a small PHP application querying the database for the routes available to us, and drawing a HTML table with toggle buttons attached.
An options panel allows us to toggle on and off overlay information for each sensor. These overlays show the ID and value of the sensor and the timestamp the data was recorded at. These overlays are switched off by default.

A slider control exists to scroll through historic data. By dragging this control, the timestamp indicated in the UI shows the period being changed to. While sliding, the map updates dynamically with traffic information for the time period indicated. To scale through historic information requires no additional API call – the information is stored in the JSON object client side on the users browser.

The UI has a legend to describe the percentage of saturation indicated by colours on the map. These colours had to be carefully chosen to avoid a clash with the large variety of colours used to indicate various features on a Google map. The Google Map is also de-saturated 30% so the colours of the map are easily visible.

The segments indicated on the map are:

- 80-100%, purple, very heavily congested.
- 60-80%, pink, heavy congestion.
40-60%, dark blue, little congestion.
0-40%, light blue, no congestion.

It was determined that having five degrees of saturation would be excessive, as the range between 0% and 40% was so insignificant in the indication of a congested roadway.

Figure 19 - Congestion Colour Legend

Figure 20 - Traffic Loop UI
We will now examine the files that make up the frontend of the application:

**app.js** – The main JavaScript frontend application.

**cityStore.php** – performs a database retrieval from the cities database, returns JSON.

**css/styles.css** – the main stylesheet for the application.

**css/mobile.css** – a mobile override to ensure the application displays on mobile devices.

**images/guide.png** – The congestion scale displayed on the right side of the user interface.

**index.php** – The structuring HTML of the user interface, and script includes to the JQuery Library and Google Maps API.

**jsonify.js** – A supporting method written by Douglas Crockford to return a JSON string representing an object. Used to cache routes in the database.

**routesTable.php** – draws a table of routes as retrieved from the database, with function calls to toggle these on the map.

**routeStore.php** – stores and retrieves JSON representations of route lines in the database.

### 4.2. Sequence Diagram

A sequence diagram what occurs on a typical page load has been modelled. Due to the size of this diagram, it has been printed in landscape format and attached under Appendix A located at the end of this document.
5. Documentation

In this chapter, we detail the steps required to set up a working copy of the Traffic Loop framework. This consists of three main conceptual steps – running the UTC Proxy, uploading and configuring the PHP application and frontend, and creating the database tables required.

5.1. The UTC Proxy

The UTC proxy is essentially a web server written in python. It’s purpose is to be polled using HTTP GET across the Internet by the PHP application.

Because this program essentially runs a web server, this requires port number 80 be forwarded to the host computer. Binding this particular port requires the python application to be run with elevated privileges: as the sudo user on *nix, or as Administrator on Windows. The following instructions describe the setup of this python program on a small home/business network – the setup on a large corporate network is beyond the scope of discussion of this report.

Port forwarding is typically achieved by logging into the configuration panel of the network’s router in a web browser (often http://192.168.0.1), and selecting port forwarding. Port number 80 must be forwarded to the IP address of the machine running the Python Web Server.

Now, the configuration options of the Python Webserver must be set. Edit config.py, updating the entry for hostname to match the IP address of the host computer.

UTC modules are placed in the /modules directory – the SCOOT module is included by default, and the additional modules is described in the section entitled ‘Writing a UTC Proxy Module in Python’.

Once the UTC modules have been gathered and the required ports forwarded, the server can be started by executing the following command:
sudo python server.py

On windows, the server is started by running an elevated administrative command prompt and executing:

python server.py

To test the server is operating as expected, visit your external IP address in a web browser. A page should be returned with a series of traffic entries. Make note of this external IP address – we will need it later.

If no page is displayed, check port forwarding settings – often, a router needs to be restarted before forwarding takes effect.

5.2. Database Setup

The database import file is located in /DB/trafficloop.sql. This SQL file is to be imported into an empty MySQL Table called ‘trafficloop’.

The database is seeded with default UI options, default cities and some default sensors and routes. These can all be deleted by using the administrative interface detailed below.

5.3. The PHP Application & Frontend

The PHP application can run on any externally accessible web server. First, the configuration file needs to be edited. The database username and password must be specified, along with the hostname of the database server. In addition, the external IP address of the UTC proxy must also be specified as noted above.

Now, upload all of the contents of the /php directory (including the /admin subdirectory) using FTP to the web server.

Optional: Once uploaded, the /admin directory must be secured using a .htaccess and .htpasswd file. A sample file is provided in this directory named htaccess.txt and htpasswd.txt. Edit htaccess.txt updating the path to the htpasswd file for your server. Rename these files after uploading to the server to ‘.htaccess’ and ‘.htpasswd’. After renaming the files may disappear. This creates an administrative username of ‘admin’ with the password ‘tr@fficl00p’.
Take note of the hostname of the web server being used – from herein, we will refer to this hostname as tlhostname.com. Replace this with the IP address or domain of your webserver.

To verify the PHP application has been installed correctly, try requesting traffic information from the application. Visit tlhostname.com/retrieve.php – this should return all the traffic information provided by the UTC proxy, and fill the database with initial traffic values.

If traffic information is not returned, examine the errors presented by the PHP parser. Likely problem areas include database configuration and an incorrect UTC proxy hostname.

Finally, if the PHP API works as expectedly all that is left is to authorize the use of Google Maps on your new domain. Visit http://code.google.com/apis/maps/signup.html and enter the URL of your webserver (e.g. tlhostname.com) into the box shown. Click generate API key, a key should be returned similar to the following:

ABQIAAAAssW_vpKQ26cxNvwYJwawxhT0mLCFgy2urHL3RJMCUjetNV5GcBQGv6jcL08S8bkWHzTZk6AdJvFQYog

Edit the Google Maps script include of index.php with this new API key:

```html
<script src="http://maps.google.com/maps/api/js?sensor=true&amp;key=ABQIAAAAssW_vpKQ26cxNvwYJwawxhT0mLCFgy2urHL3RJMCUjetNV5GcBQGv6jcL08S8bkWhTZk6AdJvFQYog" type="text/javascript"></script>
```

Now test the frontend – visit tlhostname.com and the AJAX Google maps frontend should appear with traffic information. Try toggling routes on and off, and displaying the sensor markers under ‘options’.

### 5.4. Writing a UTC Proxy Module in Python

The goal of a UTC Proxy Module is to retrieve traffic data from a set of sensors tied to a particular UTC or traffic monitoring system. The module is written in Python, and an boilerplate code (Sample.py) is supplied.

The sample code, and all additional modules go in the ‘Modules’ directory of the Python server. To create a new module, duplicate Sample.py renaming to the module name of your choice – remember this name throughout. For the sake of example, we will call the module ‘NewUTC.py’.

Edit the file ‘NewUTC.py’, renaming ‘Sample’ to ‘NewUTC’.
Now, write some Python code to retrieve a group of sensors. All code is to be written in the ‘getSensors’ function definition, however feel free to create some supporting functions as required. Note that ‘Sensor’ is a Python object that is created with the following syntax:

```python
```

It is an array of these Sensor objects that is expected as a return value of the module. For a more extensive piece of sample module code, see Scoot.py.

5.5. Adding your city

The first step to adding a new city is to examine the existing sensor architecture that will be utilised. Is there just one UTC which data will be taken from, or will sensor input come from multiple sources? Either way, a UTC Proxy Module must be written and installed for each input source, as detailed above.

Before beginning, it is recommended you have a list of sensors available and their positions to hand. Use of a mapping system capable of returning GPS coordinates (E.g. Google Maps in Google Chrome) would also be beneficial.

Now that you’ve verified your cities UTC system has a module written, it’s time to visit the administrative panel of the PHP application. This resides at:

http://www.tlhostname.com/admin

You may be prompted for a password if you renamed the htaccess.txt file as outlined above.

5.5.1. Cities Panel

The first panel you are presented with is the ‘Edit cities’ panel. Here, you are presented with a list of cities, their latitude and longitude position and the zoom level. To add a city, use the form on the left hand side of the page. Enter the city name (e.g. Cork), the latitude (e.g. 51.8972), the longitude (e.g. -8.46037) and the level of zoom. Level 14 is recommended, unless the location being added is a particularly small town. Click ‘Insert’ to add the city.
5.5.2. Routes Panel

After adding cities, the next step is to plot the routes that the system will be mapping. Each route should be describable by a Name (e.g. N8), a direction (e.g. inbound) and a list of unique sensors that make up the route. Click on the ‘routes’ tab, and a list of routes in the system along with their direction and list of sensors is presented.

To add a route, enter its name and direction. Inbound is in the direction of the city centre, outbound is leaving the city centre. For ring roads or peripheral routes, direction is unimportant – allocate arbitrarily one direction to be inbound and another outbound.

The last thing required for a route is the list of sensors. Choose carefully, as these sensors will be used to plot the route line on the map. They should be input in order, in the direction of travel. If the sensors are ordered incorrectly, the route will not plot properly.
5.5.3. Sensors Panel

The final step is to add the sensor position mappings. Insert each sensor by ID, latitude and longitude position. It is important that the latitude and longitude positions are accurate, and map the correct lane / side of the roadway. Only the sensors required by the routes mapped could be added, or alternatively every traffic sensor that exists in the city and can be returned by the UTC Proxy could be mapped for the sake of future proofing the system.

![Sensors Panel](image)

**Administration**

**Edit Sensors**

<table>
<thead>
<tr>
<th>Sensor ID</th>
<th>Latitude</th>
<th>Longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC120A1</td>
<td>51.9027</td>
<td>-8.45032</td>
</tr>
<tr>
<td>DC121A1</td>
<td>51.9002</td>
<td>-8.46306</td>
</tr>
<tr>
<td>DC123A1</td>
<td>51.9004</td>
<td>-8.46441</td>
</tr>
<tr>
<td>DC124A1</td>
<td>51.9015</td>
<td>-8.46381</td>
</tr>
</tbody>
</table>

Figure 24 – Edit Sensors Panel

5.5.4. Options Panel

Only two options exist. First, the refresh rate of the PHP application in polling the UTC proxy can be set. This defaults to 3 minutes, but could be altered to a more or less frequent poll if required.

Secondly, normalisation over time can be disabled. If the sensor type being used is shown not to return inaccurate spikes and tends to be accurate at all times, this feature would best be disabled.
5.6. Troubleshooting

What follows are some common problems that may be encountered in the setting up of Traffic Loop:

**ImportError: No module named json** when running the UTC Proxy

Upgrade to the latest version of Python, which includes the JSON module by default. Alternatively, include the JSON module in the same directory as server.py.

**The PHP Application returns an error**

Analyse the errors that are shown. The most popular stumbling block is database connection errors and incorrect timestamps being returned by proxy modules.

**Most / All routes are within 40-60% (navy colour) traffic**

If using the SCOOT UTC module for testing, with simulation mode turned on the traffic values returned are randomised. By default, the PHP Application normalises over the ten most recent values in a 15-minute interval. If these 15 values are completely random, the averaging operation has a high propensity to return a value between the 40% to 60% range. For more interesting simulations, disable normalisation over time using the administrative panel.
6. Evaluation

In this chapter, we evaluate the framework under the headings of portability, performance and possible applications of Traffic Loop. Finally, we will examine the expansion possibilities of Traffic Loop.

Providing a traffic-mapping system that functions is not a difficult task. Providing something that can truly function as a framework, however, was the goal of this project. By evaluating performance and portability, we prove that this project can truly be considered a framework.

6.1. Portability

The three-tier nature of the architecture aims to provide capacity for a very large degree of portability. First and foremost, the proxy proves a useful means of taking information that is on a file system, and making it possible to request this information through an API. The proxy can also request information from other APIs, aggregating data and returning it all in a consistent format.

The information returned by the UTC proxy alone is probably not much use without the addition of further info about the sensors being described. This is part of the function of the PHP application. The data returned by the JSON of the PHP application encodes either a single, or a whole range of historic traffic information for a variety of routes.

Designing another UTC module was outside of the scope of goals of this project, and as a result the true portability of the system has not been tested on a real-world working system. However, based on the examination of abstraction outlined in the design chapter we can conclude that in theory the system is easily portable to alternate architectures.
6.2. Performance

Throughout development, one of the major considerations architecting a solution to the problem of congestion mapping was that the system performs. To this end, calls to the UTC Proxy from the PHP application have been minimised to reduce load on the UTC system.

The JavaScript has been optimised to reduce the number of requests it needs to make. Each link between two sensors being drawn on the map is represented as a Google Polyline. These polylines are cached in the database to overcome the need to poll Google’s direction service for every subsection of the route. Instead, every route required is returned in one request as a JSON object.

The Yahoo YSlow tool measures performance of a web page based on a set of 34 rules including minimising HTTP requests, the caching setup of the server and avoiding redirects. The JavaScript was analysed by the Yahoo YSlow rating, and achieved a grade ‘A’ rating with an overall score of 91.

Page load times averaged at 1.6 seconds, and the API typically returns within 0.7 seconds even when running on a shared hosting environment.

6.3. Benefits of Traffic Loop

We will now examine the benefits of Traffic Loop, and how this system could potentially be useful to individuals in various walks of life.

6.3.1. Academics

Traffic Loop provides a very useful real time monitoring facility of traffic information, but the PHP application beneath this is also useful as an API. The API could be polled for historic data that could be used to study the effects of freak weather incidents, traffic accidents and more.

Researchers in the University of Washington found the use of a traffic mapping system very useful when examining the effect of a snowstorm on the traffic of a city. Traffic loop can potentially expand upon this. Because there is no tie to one particular monitoring solution or sensor type, the degree of information available to examine is greater.
6.3.2. Traffic Controllers

Traffic controllers are typically responsible for ensuring that cities traffic runs smoothly throughout the day. Although many US cities have the ability to view information relating to real-time traffic volumes on a map based display, the same could not be found in many European cities. Traffic Loop could give the ability to traffic controllers to have an easy to use picture of the real-time traffic situation along with the ability to view historic data. Traffic Loop could assist in making educated decisions about light sequence times. Traffic Loop could potentially help in identifying problem areas, and enable controllers to take corrective action before the congestion intensifies.

Because the system is based on Google Maps, there are no constraints on the resolution of the map – the web page is fluid width and height. When deployed on a large resolution monitor, it would be possible to view the majority, if not all routes that are covered within a city.

6.3.3. Emergency Services

Emergency Services controllers could utilise Traffic Loop to guide vehicles across the city in rush hour, or even to avoid unexpected congestion that can often occur because of an event in an area of the city. Guiding emergency vehicles through the least congested route enables vehicles to reach the scene of an emergency faster.

6.3.4. Hauliers

Hauliers rely on the countries road network to get goods from their source to destination, and their ability to do so is vital to Ireland’s status as an ideal base for foreign trade amid multinationals. Typically, a haulier has a deadline to meet, such as the departure time of a ferry or air transport. Missing this deadline can potentially cause delays to production or to the delivery of finished products, damaging Ireland’s reputation as a country of choice for foreign trade. The ability to view congestion information and adapt a route in real time could save hauliers a great deal of time that would normally be spent in congestion.
6.3.5. Public Transport Operators

As taxi drivers set out towards their destination, it would be advantageous to know where congestion exists. This reduces journey time and cost for the customer, and enables the taxi driver to get to his next fare quicker.

Bus routes typically travel along congested, busy routes in and out of town. Whilst bus lanes are now commonplace on these routes, congestion can still have an effect. Being able to see this congestion information enables drivers to inform their passengers of potential delays, and route schedulers can plan ahead of time.

6.3.6. Commuters

Commuters travelling by bus can often face unexpected delays due to high volumes of traffic along their route. By knowing in advance of such delays, commuters can compensate, alert their colleagues to their delay or even plan to use an alternative bus route.

Although the practice of bringing a private vehicle into a city centre is a practice to be discouraged when viable public transport alternatives exist, it can occasionally be unavoidable. Making traffic congestion information available to commuters can help to plan alternative less congested routes, reducing journey times and commuter stress\(^{45}\).

6.3.7. Environmental Benefits

Under every subheading examined so far, the goal of Traffic Loop has been to reduce journey times. The likely approach to doing this is to encourage cars to utilise less congested routes for their journey. An automobile is far less efficient sitting in traffic with the engine running for significant lengths of time, emitting unnecessary emissions. The International Road Transport Union recently found that traffic congestion increases CO2 emissions by 300\(^{46}\). Traffic Loop could potentially greatly reduce the length of time automobiles sit in traffic, which would have environmental benefits.
7. Conclusions

7.1. Expanding upon Traffic Loop

As we’ve already seen, traffic loop has a very portable and extensible architecture. Since the PHP application acts as an API and is abstract from the frontend, the construction of a whole number of other traffic related applications is possible. By polling the API, below are some suggestions on ways to expand upon Traffic Loop.

7.1.1. Twitter Sentiment Analysis

Sentiment Analysis of social media, Twitter in particular, is in its infancy. Researchers at Stanford University have found that using Twitter as a source for user sentiment is a viable approach\(^\text{47}\).

Twitter has also proven reliable at predicting the outcome of many voting based events, such as the outcome of TV show phone in voting\(^\text{48}\).

As the technology develops, it may become possible to parse the twitter public stream\(^\text{49}\) to determine how traffic is progressing across a route, which could be used as an input to a Traffic Loop mapping system.

7.1.2. Computer Vision

The UTC Proxy lends itself well to Computer Vision based approaches to traffic monitoring, and Traffic Loop could prove an ideal platform to make the transition from induction loop based measurement to a computer vision approach. Migration could be performed on a route-by-route basis, benchmarking one system against the other.
7.1.3. Public Transport Probes

Using methods of Public Transport as a congestion probe (Fagan, Meir. Proceedings of ITRN2010, 2010)\textsuperscript{50} has proven successful as a means of measuring real time congestion across routes. Although the use of busses does have implications (bus lanes typically flow reasonably freely even when the main traffic lane is congested), this type of monitoring could still prove valuable as an input source for Traffic Loop.

7.1.4. Data Visualisation

Being able to poll historic data makes traffic loop invaluable for data visualisation. Historic values could be plotted on informative grids to show how traffic evolves across the city throughout the day, week or even year.

7.1.5. Native Mobile Application

The API also makes for very easy implementation of a mobile application, using the iPhone or Android SDK. The information returned from a POST requesting route information is sufficient to plot any route on a map that accepts GPS coordinates as waypoints. Although a web application already exists that works on mobile devices, harnessing the power of native application might be beneficial. Traffic Loop vastly reduces the cost of developing such an application.

7.2. Criticisms

Although this project has been highly successful in producing a traffic mapping solution, it is not without its flaws.

Firstly, the nature of the abstraction layer purposely hides the source of the information from the database and the end user. It could be argued that in the event of unreliable data propagating through the system, it would be beneficial to have the ability to store in the database the source of the sensor (traffic loop, camera, etc.). This was eventually decided as unnecessary, because the unique ID of the sensor can be referenced back to the UTC Proxy, where it’s type will be discovered. In addition to storing the source, it would perhaps have
been beneficial to store the un-normalised value that was taken directly from the sensor at the time of recording. Since this value is not stored anywhere, it is lost.

Another potential criticism is that the system does not allow for multiple sources of input across overlapping sections of a route. This means that for one line segment of congestion being drawn on a map, only two unique sensors are being considered. There is no ability to take a secondary or weighted input from another source along the route. The reason for not considering such secondary input was the difficulty of representing such information in a meaningful mapping in a database or software system. If one secondary input is allowed, why not allow multiple? If a secondary input is needed, surely this suggests the primary two sensors mapping the route are flawed to begin with, and should be replaced with a more accurate alternative?

There are strong arguments for and against taking more than two sensor inputs to map congestion along a segment of a route.

7.3. Conclusions

Thus far we have seen that the Traffic Loop system is potentially quite portable, and has capacity to run on one or more architectures. The system can also be deployed in any country in which the Google Direction Service operates (any country where you can plot a series of directions using Google Maps). Even if this direction service does not exist, the frontend could easily be modified to draw straight lines between sensor coordinates, a less aesthetically elegant solution but functional none the less.

We have also seen that the system can perform to a high standard. On a large deployment, with high volumes of traffic this system will scale to meet demand. A traffic monitoring solution could be prone to peaks of demand prior to beginning a commute to work and prior to the return journey home. The hardware that serves this application (the LAMP stack) lends itself well to performance tweaking under high load. Ultimately, this means Traffic Loop will remain available whatever the demand.

We have seen that the system is highly extensible. A whole host of interesting potential spinout projects have been presented as a result of this framework. Many more possibilities exist that are beyond the scope of discussion here.
Lastly, we have seen that this system can be useful to a large portion of people who interact with the world’s roadways, and would bring great benefit to many walks of life.

We have examined some minor criticisms of the system, but these are largely opinion based and do not effect the end result. A functional, highly usable system has been delivered.

In the introduction we presented a number of goals, all of which have been met. Firstly, our system is capable of taking multiple values from traffic systems, collating their data and normalising the values to a predefined 0-100% scale. This means values from differing inputs can be considered comparable, and this collated data is mapped visually without the need for on-going human input from traffic controllers.

This also means we have successfully freed up traffic controllers from performing this arduous, time intensive task, allowing them to focus on more important issues such as traffic light sequence modifications.

We have also ensured the load on the UTC is minimal – our carefully designed local replica of data ensures the UTC proxy is polled a maximum of once every three minutes, a user configurable value.

Finally, it is fair to call this system a framework, considering the potential for portability, abstraction and ease of implementation that exists. The goals set out at the beginning of this project have been met in their entirety.
8. Appendix

8.1. Appendix A: Sequence Diagram